

# COMPUTER SYSTEM ASSIGNMENT 2 REPORT

**Student Name:** Anh Tuan Doan  
**Student ID:** 103526745  
**Unit code:** COS10004  
**Lab session:** Friday 8:30am-10:30am at EN409

## Table of Contents

**Stage 1 .....2**

**Stage 2: .....2**

**Stage 3: .....3**

**Stage 4: .....4**

**Stage 5: .....5**

    Stage 5a:..... 5

    Stage 5b ..... 6

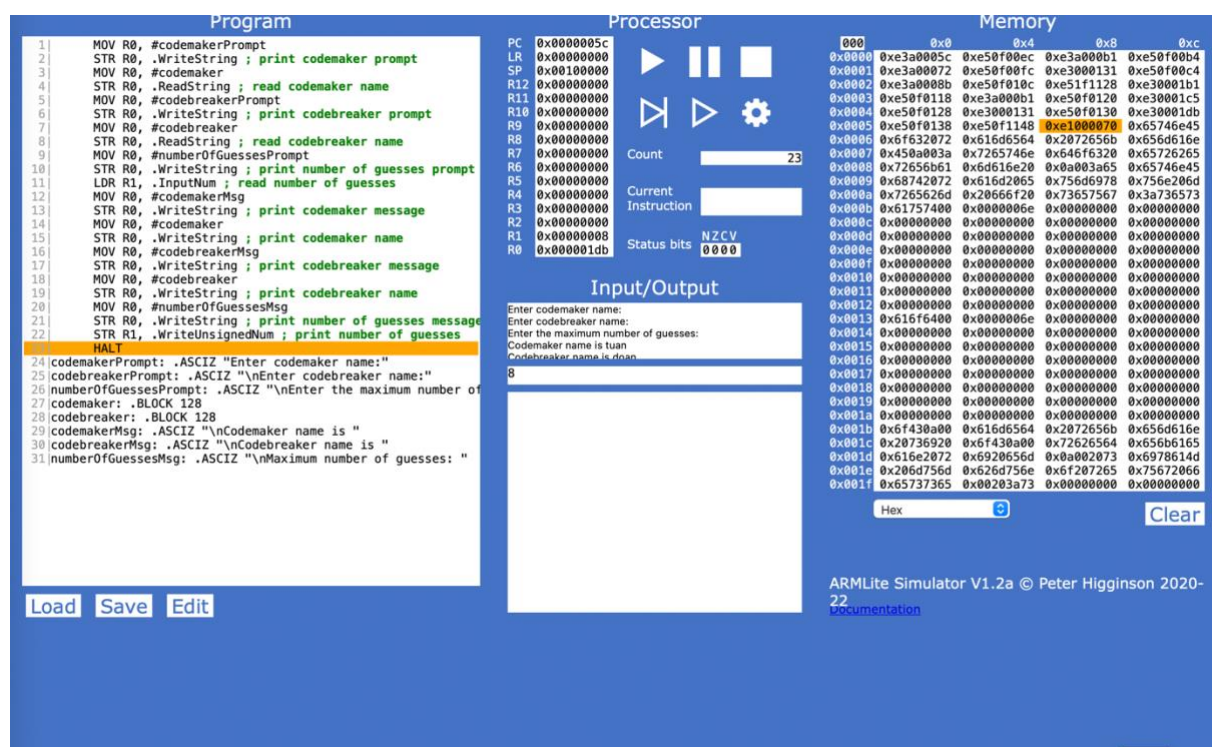
**Stage 6: .....7**

**Assumptions .....8**

**Unresolved Problem .....8**

## Stage 1

In the first stage, I created multiple labels including the prompts that ask for the user inputs as well as 2 labels to store the string of the code maker and code breaker name. To print the prompts to the screen, I moved the address of the message to Register 0 (R0) first, then store R0 in the pre-defined label called ".WriteString". To get the user input, I use ".ReadString" for strings and ".InputNum" for a number. The code breaker and code maker's names are stored in the labels "codebreaker" and "codemaker" respectively, while the maximum number of guesses is stored in R1. After receiving the user inputs, the program also prints messages to the screen to let the user know what they have entered.



## Stage 2:

In this stage, I defined a function called "getcode" that reads in 4-character colour code from the user and store it in an array. The array to store the string in is passed in as a parameter to the function. Because this function follows the ABI convention, the input will be stored in R0. Moreover, because this function uses R2, R3, and R4, I pushed those Registers on the stack before using them and pop them back when the function ends. Firstly, the function prints a prompt to ask the user to enter their code and then stores the input code in R2. The next step to do is to check if the input satisfied the requirement that each character is only one of 'r', 'g', 'b', 'y', 'p' or 'c', and the total number of characters is exactly 4. To do this, I used Indexed Memory Addressing technique to iterate through the array and check if each character is one of the allowed characters by using ASCII codes. If the character is allowed, I incremented the index and move on to the next character. However, if there is at least one character that is not 'r', 'g', 'b', 'y', 'p' or 'c', this whole process will restart and the program will ask the user for a new code. Moreover, the function also has to check the

number of characters in the input string. To do this, I compare the index with 4 every time the loop iterates. If the index is less than 4, then the loop iterates again; if the index is 4, then I check if the current character is null. If the character with the index 4 is null, it means that the code has 4 characters, otherwise, the function will restart and ask the user for a new code.

The image shows a software interface with two main panels: 'Program' and 'Processor'.

**Program Panel:** Contains assembly code for a function named 'getcode'. The code includes comments in green and instructions in black. It pushes registers R2, R3, and R4 onto the stack, moves the input from R0 to R2, and enters a loop. In the loop, it moves the address of 'codePrompt' to R3, prints the prompt, reads a string into R2, and increments an index in R3. A 'checkcode' section follows, where it loads the character at [R2 + R3] into R4 and compares it with 'r', 'g', 'b', 'y', 'p', and 'c'. If it doesn't match any, it restarts the loop. If it matches, it increments the index and checks if it's less than 4. If not, it checks the next value. If the last character is not null, it restarts the loop. Finally, it pops the registers and returns.

**Processor Panel:** Shows the state of the processor. It includes a list of registers (PC, LR, SP, R12, R11, R10, R9, R8, R7, R6, R5, R4, R3, R2, R1, R0) and their current values. The PC is 0x0000006c. The LR is 0x00000064. The SP is 0x00100000. The R12, R11, R10, R9, R8, R7, R6, R5, R4, R3, R2, R1, and R0 are all 0x00000000, except for R0 which is 0x0000025b. There are also buttons for 'Count' (195), 'Current Instruction', and 'Status bits' (NZCV: 0100).

**Input/Output Panel:** Contains a text area for 'Enter a code:' and a 'Program HALTED. STOP, LOAD or EDIT' message. Below it is a 'crgb' label and a large grid area.

At the bottom of the 'Program' panel, there are three buttons: 'Load', 'Save', and 'Edit'.

### Stage 3:

In this stage, I used the function "getcode" defined in stage 2 to get the secret code from the code maker. Firstly, I defined a label for the secret code and print out a prompt to ask the code maker to enter their secret code. Next, to call the "getcode" function, I pushed the previous value of R0 to the stack, and then move the memory address of "secretcode" to R0. Next, I called the "getcode" function using "BL getcode" command. Finally, I pop the previous value of R0 back.

### Program

```

1|  MOV R0, #codemakerPrompt
2|  STR R0, .WriteString ; print codemaker prompt
3|  MOV R0, #codemaker
4|  STR R0, .ReadString ; read codemaker name
5|  MOV R0, #codebreakerPrompt
6|  STR R0, .WriteString ; print codebreaker prompt
7|  MOV R0, #codebreaker
8|  STR R0, .ReadString ; read codebreaker name
9|  MOV R0, #numberOfGuessesPrompt
10| STR R0, .WriteString ; print number of guesses prompt
11| LDR R1, .InputNum ; read number of guesses
12| MOV R0, #codemakerMsg
13| STR R0, .WriteString ; print codemaker message
14| MOV R0, #codemaker
15| STR R0, .WriteString ; print codemaker name
16| MOV R0, #codebreakerMsg
17| STR R0, .WriteString ; print codebreaker message
18| MOV R0, #codebreaker
19| STR R0, .WriteString ; print codebreaker name
20| MOV R0, #numberOfGuessesMsg
21| STR R0, .WriteString ; print number of guesses message
22| STR R1, .WriteUnsignedNum ; print number of guesses
23| ;-----
24| MOV R0, #0x0A
25| STRB R0, .WriteChar ; print a new line character
26| MOV R0, #codemaker
27| STR R0, .WriteString ; print the codemaker name
28| MOV R0, #secretcodePrompt
29| STR R0, .WriteString
30| PUSH {R0}
31| MOV R0, #secretcode
32| BL getcode ; get the secret code
33| POP {R0}
34| HALT
35| getcode:
36| ; R0: the array that will store the code
37| PUSH {R2,R3,R4} ; are we allowed to use R0,1,2,3 if our function does
38| MOV R2, R0 ; copy input of R0 to R2
39| loop:
40| MOV R3, #codePrompt

```

Load

Save

Edit

### Processor

PC	0x00000084
LR	0x0000007c
SP	0x00100000
R12	0x00000000
R11	0x00000000
R10	0x00000000
R9	0x00000000
R8	0x00000000
R7	0x00000000
R6	0x00000000
R5	0x00000000
R4	0x00000000
R3	0x00000000
R2	0x00000000
R1	0x00000009
R0	0x000002a0

Count

84

Current Instruction

Status bits

NZCV 0100

### Input/Output

Codebreaker name is doan

Maximum number of guesses: 9

uan, please enter a 4-character secret code

Enter a code:

Program HALTED. STOP, LOAD or EDIT

yprg

## Stage 4:

In the next stage, the program allows the code breaker to enter their guesses. Firstly, I defined an array "querycode" that will store the code breaker's guesses. Next, the program prints out some messages to ask the user to enter the guess and call the "getcode" function to get the user input in a similar way to stage 3. Notably, the maximum number of guesses is currently stored in R1 back in stage 1. Each time the codebreaker enters a code, the program will decrease R1 by 1 and compare R1 with 0. If R1 is not equal is 0, the program will ask for a new code again. If R1 is 0 then it means that the user has entered the maximum number of guesses and the program will stop.

The screenshot shows a debugger window with assembly code on the left and a register/status panel on the right. The assembly code is in ARMv7-M syntax. The register panel shows the PC at 0x000000bc and various other registers. The status panel shows the Count at 372 and the Status bits as 0100.

```

24  STRB R0, .WriteChar ; print a new line character
25  MOV R0, #codemaker
26  STR R0, .WriteString ; print the codemaker name
27  MOV R0, #secretcodePrompt
28  STR R0, .WriteString
29  PUSH {R0}
30  MOV R0, #secretcode
31  BL getcode ; get the secret code
32  POP {R0}
33 askQuerycode:
34  MOV R0, #0x0A
35  STRB R0, .WriteChar ; print a new line character
36  MOV R0, #codebreaker
37  STR R0, .WriteString ; print the codebreaker name
38  MOV R0, #querycodePrompt
39  STR R0, .WriteString ; print the query code prompt
40  STR R1, .WriteUnsignedNum ; print the current guess number
41  PUSH {R0}
42  MOV R0, #querycode
43  BL getcode ; get the query code
44  POP {R0}
45  SUB R1, R1, #1
46  CMP R1, #0 ; Compare the current guess number with 0
47  BNE askQuerycode
48  HALT
49 getcode:
50 ; R0: the array that will store the code
51 PUSH {R2,R3,R4}
52 MOV R2, R0 ; copy input of R0 to R2
53 loop:
54 MOV R3, #codePrompt
55 STR R3, .WriteString ; print the code prompt
56 STR R2, .ReadString ; read the string
57 MOV R3, #0 ; index
58 checkcode:
59 LDRB R4, [R2 + R3]
60 CMP R4, #0x72 ; compare R4 with r
61 BEQ cont
62 CMP R4, #0x67 ; compare R4 with g
63 BEQ cont

```

Register values (R15 to R0):

- PC: 0x000000bc
- LR: 0x000000a8
- SP: 0x00100000
- R12: 0x00000000
- R11: 0x00000000
- R10: 0x00000000
- R9: 0x00000000
- R8: 0x00000000
- R7: 0x00000000
- R6: 0x00000000
- R5: 0x00000000
- R4: 0x00000000
- R3: 0x00000000
- R2: 0x00000000
- R1: 0x00000000
- R0: 0x00000401

Status bits: NZCV 0100

Count: 372

Current Instruction:   
doan, this is guess number: 2  
Enter a code:  
doan, this is guess number: 1  
Enter a code:  
Program HALTED. STOP, LOAD or EDIT

Input/Output:   
cybg

Buttons: Load Save Edit

## Stage 5:

### Stage 5a:

In the next stage, the program compares the secret code and the query code. The parameters will be stored in R0 and R1 according to the ABI convention. The functions also push the registers that it will use before changing them and pop the values back before the function terminates. There are 3 cases that may occur:

- Case 1: the query peg is a direct match with its corresponding peg in the secret code, OR
- Case 2: the query peg is a match with at least one other peg in the secret code but not in the correct position, OR
- Case 3: the peg has no match in the secret code

This function will return the number of case 1 in R0, and the number of case 2 in R1. To find the number of case 1 and 2, I iterate through the secret code and for each peg I check if the query code has the same colour at the exact position, if that's true then I increment case 1 by 1, otherwise, I iterate through the "querycode" to find if the querycode has that colour but in a different location and increment case 2 by 1 if that's true. If there is no match in the "querycode", I move on to the next character in the secret code. When the index for the secret code reaches 4, the function will terminate.



```

90|comparecodes:
91|;R0 secretcode, R1 querycode
92|    PUSH {R2,R3,R4,R5,R6,R7}
93|    MOV R2, R0          ; copy secretcode to R2
94|    MOV R3, R1          ; copy querycode to R3
95|    MOV R0, #0          ; case 1
96|    MOV R1, #0          ; case 2
97|    MOV R4, #0          ; index for secret code
98|iterateSecretcode:
99|    LDRB R5, [R2 + R4] ; get secretcode peg at index
100|    LDRB R6, [R3 + R4] ; get querycode peg at index
101|    CMP R5, R6          ; check the pegs
102|    BNE else
103|    ADD R0, R0, #1      ; increment case 1
104|    B incrementSecretcodeIndex
105|else:
106|    MOV R7, #0          ; index for query code
107|iterateQuerycode:
108|    LDRB R6, [R3 + R7] ; get querycode peg at index
109|    CMP R5, R6
110|    BNE incrementQuerycodeIndex
111|    ADD R1, R1, #1      ; increment case 2
112|    B incrementSecretcodeIndex
113|incrementQuerycodeIndex:
114|    ADD R7, R7, #1
115|    CMP R7, #4
116|    BNE iterateQuerycode
117|incrementSecretcodeIndex:
118|    ADD R4, R4, #1
119|    CMP R4, #4
120|    BNE iterateSecretcode ; stop if index reaches 4 (should we check it wi
121|    POP {R2,R3,R4,R5,R6,R7}
122|    RET
123|codemakerPrompt:    ASCII "Enter codemaker name:"

```

## Stage 5b

Finally, using the returned value of the "comparecodes" function, I print a message to let the user know how many position matches (case 1) and colour matches (cases 2) there are. Moreover, every time the user enters their guess, I check the position matches to see if it is 4. If that is true, I print a message "you WIN" to the screen and end the program with a "Game Over" message. If the position matches are not 4, I checked the current guess number stored in R1. If the current guess number is 0, then the codebreaker loses the game and a "Game Over" message is also printed out on the screen:

0x00088

Program

```

34 askQuerycode:
35     MOV R0,#0x0A
36     STRB R0,.WriteChar ; print a new line character
37     MOV R0,#codebreaker
38     STR R0,.WriteString ; print the codebreaker name
39     MOV R0,#querycodePrompt
40     STR R0,.WriteString ; print the query code prompt
41     STR R1,.WriteUnsignedNum ; print the current guess number
42     PUSH {R0}
43     MOV R0,#querycode
44     BL getcode ; get the query code
45     POP {R0}
46     PUSH {R0, R1}
47     MOV R0,#secretcode
48     MOV R1,#querycode
49     BL comparecodes
50     MOV R3,R0 ;case1
51     MOV R4,R1 ;case2
52     POP {R0,R1}
53     MOV R0,#0x0A
54     STRB R0,.WriteChar ; print a new line character
55     MOV R0,#case1Msg
56     STR R0,.WriteString
57     STR R3,.WriteUnsignedNum
58     MOV R0,#case2Msg
59     STR R0,.WriteString
60     STR R4,.WriteUnsignedNum
61     CMP R3,#4 ;check if case 1 is 4
62     BNE checkGuessNumber
63     MOV R0,#0x0A
64     STRB R0,.WriteChar ; print a new line character
65     MOV R0,#codebreaker
66     STR R0,.WriteString ; print codebreaker name
67     MOV R0,#codebreakerWinMsg
68     STR R0,.WriteString ; print you win message
69     B gameover
70 checkGuessNumber:
71     SUB R1,R1,#1
72     CMP R1,#0 ; Compare the current guess number with 0
73     BNE askQuerycode

```

Load

Save

Edit

Processor

PC	0x0000013c
LR	0x000000bc
SP	0x00100000
R12	0x00000000
R11	0x00000000
R10	0x00000000
R9	0x00000000
R8	0x00000000
R7	0x00000000
R6	0x00000000
R5	0x00000000
R4	0x00000000
R3	0x00000004
R2	0x00000000
R1	0x00000003
R0	0x0000053f

▶

⏸

■

⏮

⏭

⚙

Count

385

Current Instruction

Status bits

NZCV 0110

Input/Output

Enter a code:

Position matches: 4 , Colour matches: 0

doan, you WIN!

Game Over!

Program HALTED. STOP, LOAD or EDIT

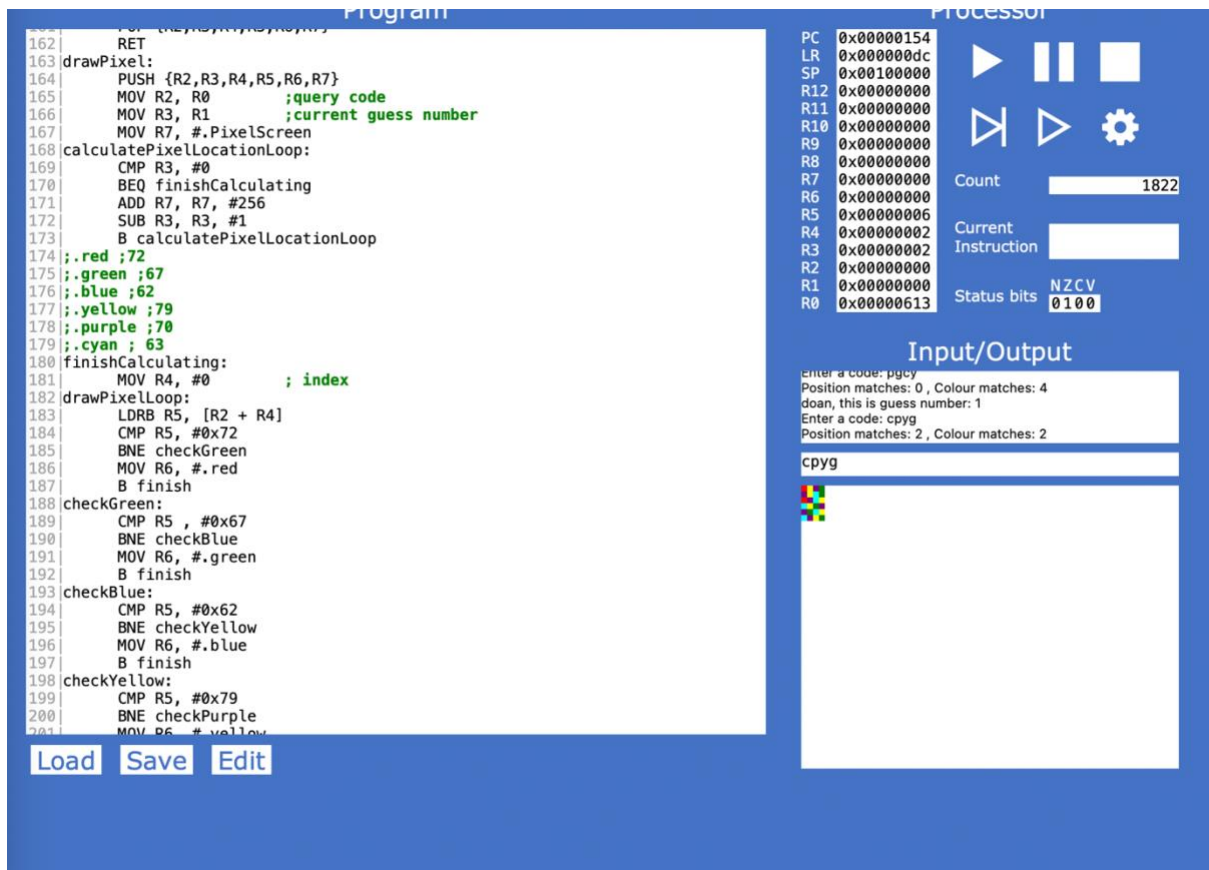
cypg

## Stage 6:

In this stage, every time the codebreaker enters their guess, a 4-pixel horizontal line will be drawn where each pixel represents the colour in the query code. To do this, I made a new function called drawPixel that accepts the query code and the current guess number as the parameters. Notably, the current guess number is not the number of guesses left. For example, if there are 4 out of 6 guesses left then the current guess number is 2. To calculate the current guess number, I subtract the number of guesses left from the total maximum number of guesses and pass it to the function.

The first thing that the function do is to calculate the location for the start of the line that the 4 pixels will be drawn. This can be calculated by this formula:  $\text{location} = \text{.PixelScreen} + 256 * N$  (where `.PixelScreen` is the location of the top-left pixel and `N` is the current number of guesses). I multiply 256 with `N` because there is 64 pixel in a row and each pixel is 4 bytes. To calculate the location, I use a loop and repeatedly add 256 to `".PixelScreen"` and subtract the current number of guesses until it reaches 0.

The next step is to draw the colour. To do this, I iterate through the pegs in the query code and move the corresponding colour to a register. Finally, I store the colour in the location that I previously calculated and increment the location by 4 for the next colour. I also checked the index when it reaches 4 to terminate the function.



## Assumptions

The functionalities are implemented based on the requirements and there are no assumptions have been made.

## Unresolved Problem

There is no unresolved problem in this program. All functionalities have been tested and the program works according to the requirements.